UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación (EUR-ACE®)

TRABAJO FIN DE GRADO

Clasificación de imágenes con redes neuronales profundas mediante conjuntos de entrenamiento reducidos y aprendizaje "few-shot"

Guillermo Eliseo Torres Alonso
Tutor: Miguel Ángel García García

JUNIO 2019

Clasificación de imágenes con redes neuronales profundas mediante conjuntos de entrenamiento reducidos y aprendizaje "few-shot"

AUTOR: Guillermo Eliseo Torres Alonso
TUTOR: Miguel Ángel García García

Video Processing and Understanding Lab
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2019

Trabajo parcialmente financiado por el Ministerio de Economía, Industria y Competitividad del Gobierno de España bajo el proyecto TEC2017-88169-R (MobiNetVideo) (2018-2020)

Resumen (castellano)

Este Trabajo Fin de Grado ha consistido en la implementación y evaluación experimental de una red neuronal profunda que permite la clasificación supervisada de imágenes mediante una serie limitada de entrenamiento. Esta red fue descrita por F. Sung, Y. Yang y L. Zhang, T. Xiang en un artículo [1] publicado en 2018 en una de las conferencias internacionales más prestigiosas en Visión por Computador y Reconocimiento de Patrones (IEEE CVPR 2018).

A diferencia del aprendizaje convencional, que emplea millones de imágenes etiquetadas, el aprendizaje "few-shot" pretende clasificar imágenes a partir de conjuntos reducidos de entrenamiento.

En este trabajo se ha implementado la red neuronal descrita en el artículo de referencia y se ha evaluado su rendimiento utilizando un conjunto de imágenes adquiridas en el campus de la UAM.

Palabras clave (castellano)

Red neuronal convolucional, entrenamiento "few-shot", clasificación de imágenes.

Abstract (English)

This Bachelor Thesis has consisted in the implementation and experimental evaluation of a deep neural network that allows the supervised classification of images through reduced training sets. This network was described in a paper [1] published by F. Sung, Y. Yang y L. Zhang, T. Xiang in 2018 in one of the most prestigious international conferences on Computer Vision and Pattern Recognition (IEEE CVPR 2018). Differently to conventional training, which uses millions of labelled images, “few-shot” learning aims at classifying images from reduced training sets.

In this work, we have implemented the neural network described in the reference paper and we have evaluated its performance with respect to a set of images acquired over the UAM campus.

Keywords (inglés)

Convolutional neural network, “few-shot” training, image classification.

Agradecimientos

A todos los profesores que me han allanado el camino hasta aquí y a mi tutor por el esmero puesto.

INDICE DE CONTENIDOS

Resumen iv

Abstract vi

Agradecimientos viii

1 Introducción **1**
1.1 Motivación 1
1.2 Objetivos 2
1.3 Organización de la memoria 2

2 Estado del Arte de las RNA **5**
2.1 Introducción 5
2.2 Redes Neuronales Artificiales 5
2.2.1 Retropropagación 9
2.2.2 Red Neuronal Convolucional 9

3 Diseño **13**
3.1 Introducción 13
3.2 Definición del programa a realizar 13
3.3 Dataset 15
3.4 Herramientas de Programación 15

4 Desarrollo **17**
4.1 Introducción 17
4.2 Desarrollo de la Red Neuronal 17
4.2.1 netEncoder 20
4.2.2 netRelation 20
4.3 Cuerpo 22
4.3.1 Fase de Entrenamiento 22
4.3.2 Fase de Pruebas 26

5 Pruebas y resultados **27**
5.1 Introducción a las pruebas y a los datasets utilizados 27
5.2 Pruebas con Datasets 28
5.2.1 Resultados de las pruebas para el dataset "miniImagenet 1" 28
5.2.2 Resultados de las pruebas para el dataset "Edificios UAM" 32
5.3 Conclusiones de las pruebas realizadas 39

6 Conclusiones y trabajo futuro **41**
6.1 Conclusiones 41
6.2 Trabajo futuro 41

Referencias **43**

Glosario **1**

INDICE DE FIGURAS

2.1. Representación del perceptrón 7

2.2. Función de regresión logística sigmoidal 8

2.3. Ejemplo de CNN, Cifar10 11

3.1. Estructura del programa para 5-way 1-shot 14

3.2. Set de imágenes de muestra y consulta 15

4.1. Estructura de toda la red neuronal y el bloque convolucional 17

4.2. Ejemplo de obtención de los mapas de características mediante la operación de Convolución 18

4.3. Resultados relacionales para cada imagen de consulta de entrada 22

4.4. Fórmula ECM 23

4.5. Representación de la función ECM 24

4.6. Ejemplos de funcionamiento de la optimización del descenso por gradiente en 2D 25

4.7. Visualización en 3D del problema a resolver mediante el algoritmo de descenso por gradiente 26

5.1. Interacción de una imagen de muestra con las 15 imágenes de consulta de la clase cangrejo 29

5.2. Imágenes de consulta clasificadas erróneamente como clase cangrejo 30

5.3. Interacción de una imagen de muestra con imágenes de consulta de clase perro 1 labrador 31

5.4. Imágenes de consulta clasificadas erróneamente como clase labrador 32

5.5. Interacción de una imagen de muestra con imágenes de consulta de clase rectorado 32

5.6. Imágenes de consulta clasificadas erróneamente como clase rectorado 33

5.7. Interacción de una imagen de muestra con imágenes de consulta de clase plaza mayor 34

5.8. Imágenes de consulta clasificadas erróneamente como clase plaza mayor 35

5.9. Interacción de una imagen de muestra con imágenes de consulta de clase plaza mayor 36

5.10. Imágenes de consulta clasificadas erróneamente como clase rectorado 37

5.11. Interacción de una imagen de muestra con imágenes de consulta de clase plaza mayor 38

5.12. Imágenes de consulta clasificadas erróneamente como clase plaza mayor 38

INDICE DE TABLAS

5.1. Resultados de la RN mediante diversos datasets 28

5.2. Comparación de la RN con estudios previos 40

1 Introducción

1.1 Motivación

Las redes neuronales artificiales están cada día más presentes en el desarrollo de las tecnologías de la computación. Las redes neuronales artificiales (RNA en adelante) consisten en la implementación de operaciones matemáticas y algoritmos de aprendizaje que puedan emular a las operaciones y conexiones biológicas existentes en la corteza cerebral humana.

En la rama de Sonido e Imagen del Grado, hay un interés creciente en este tipo de aplicaciones para su uso en el reconocimiento de imágenes, ya sea para reconocimiento de entornos, como objetos, personas, animales, etc. A esto se suma el interés científico-tecnológico que hay en esta materia, no solo por grandes o pequeñas empresas, sino también por los Estados. Este subcampo se está desarrollando para dar lugar a vehículos no tripulados capaces de recorrer largas distancias, mediante el *Machine Learning*, cámaras que pueden reconocer diversos objetos en una foto o secuencia de video, ayudar a caminar a robots, el campo de *Big Data* etc.

Esta tecnología puede suponer un cambio socioeconómico inmenso, tanto para explorar nuevas vías de negocio y productos tecnológicos, como a la hora de interactuar las personas entre ellas, unida a las redes 5G que empiezan a estar operativas ya en algunas ciudades de países occidentales, e incluso en el campo de la investigación de la medicina[1].

Esta memoria de TFG es la síntesis del trabajo desarrollado de una aplicación SW basada en estas redes neuronales que agrupa los conocimientos adquiridos sobre esta materia, las pruebas realizadas y expone los resultados obtenidos.

El proyecto pretende mostrar la capacidad de reconocer imágenes clasificadas en diversas clases mediante una red RNA siguiendo el esquema propuesto por F. Sung, Y. Yang y L. Zhang, T. Xiang en un artículo de referencia [1] publicado en 2018 en una de las conferencias internacionales más prestigiosas en Visión por Computador y Reconocimiento de Patrones

[1] https://doi.org/10.1093/bioinformatics/btw255

(IEEE CVPR 2018). A lo largo de esta memoria, la propuesta se identificará simplemente como el artículo de referencia o el artículo simplemente.

Los resultados obtenidos en las pruebas se compararán con los expuestos por el artículo.

1.2 Objetivos

Los objetivos propuestos por este Trabajo de fin de grado son:

- Comprender el funcionamiento de una red neuronal artificial.
- Desarrollar el algoritmo que permita realizar el reconocimiento de imágenes por clases.
- Realizar un programa que implemente el algoritmo de red neuronal propuesto, y seguir las distintas fases de entrenamiento y pruebas.
- Comparar los resultados obtenidos con los indicados en el artículo de referencia y finalmente extraer las conclusiones pertinentes.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1:** Expone las motivaciones, los objetivos y la organización de la memoria.
- **Capítulo 2:** Nos acerca a las redes neuronales artificiales, su evolución, los métodos de desarrollo y se propone el tipo de RNA a utilizar en el TFG.
- **Capítulo 3:** Define el diseño de la red neuronal propuesta para este TFG.
- **Capítulo 4:** Presenta el desarrollo seguido para construir el programa que implementará el algoritmo de la RNA seleccionada.
- **Capítulo 5:** Analiza los resultados obtenidos tras la fase de aprendizaje y prueba por el programa y los compara con los resultados indicados en el artículo de referencia.
- **Capítulo 6:** Realiza una breve conclusión acerca de la materia y las experiencias obtenidas por el autor del proyecto.

2 Estado del Arte de las RNA

2.1 Introducción

En esta sección se introducen los conceptos necesarios para comprender el diseño de una Red Neuronal Artificial (RNA), así como su evolución y los elementos clave usados en este proyecto.

2.2 Redes Neuronales Artificiales

Las RNA, fruto del desarrollo científico y tecnológico, han hecho converger los estudios realizados sobre el funcionamiento de las neuronas de la corteza cerebral humana y el desarrollo de las Matemáticas, la Estadística, la Programación y de sus herramientas, que originaron la Inteligencia Artificial (IA).

Mediante la IA, los expertos investigan las maneras de implementar procesos inteligentes, algunas propias de los seres vivos, para que sean realizados por máquinas. Esto supuso una revolución para la tecnología y el concepto de IA tiene origen en 1956, aunque los estudios y desarrollos previos, como los de Alan Turing, dieron origen a la informática teórica y a las computadoras inteligentes [2]. Una de las características más importantes de la IA para el desarrollo de las RNA es el aprendizaje automático o Machine Learning.

El aprendizaje automático consiste en el desarrollo de algoritmos de programación aplicados a datos por máquinas para conseguir realizar ciertos problemas, que pueden ser predicciones, reconocimiento de figuras en una imagen o video, etc. Estos problemas pueden ser abordados de diferentes maneras, permitiendo un aprendizaje automático: pudiendo ser un aprendizaje supervisado, semisupervisado, no supervisado, poniendo o no etiquetas a los datos iniciales del problema para que partan de una referencia a la hora de realizar el entrenamiento; por refuerzo por el cual el programa se va retroalimentando de experiencias externas mediante ensayo-error; multitarea acumulando experiencias de situaciones previas y transducción, parecido al aprendizaje supervisado pero sin construir una función, para realizar predicciones futuras a partir de los valores de entrada. Por otro lado, el aprendizaje puede ser abordado como un problema de regresión o clasificación, siendo la regresión un modelo que busca extraer la relación de todos los resultados numéricos obtenidos a partir de los datos

iniciales y la clasificación un problema que clasifica los datos o las características extraídas de los datos en categorías.

De manera simplificada, las redes neuronales biológicas funcionan a partir de neuronas entrelazadas que forman capas de neuronas y, a partir de impulsos eléctricos, transmiten una información o datos a procesar para después elaborar una respuesta mediante mecanismos de aprendizaje. Esto permite también clasificar los impulsos eléctricos en señales de diversas naturalezas o clases para generar respuestas correctas.

Las RNA empezaron a formularse de manera simple con la finalidad de resolver problemas sencillos mediante procesos que utilizaban una sola neurona, para generar programas supervisados o no. Estos procesos estaban compuestos por el perceptrón simple como la unidad de la RNA formada por una capa de entrada y una de salida, que realizaban predicciones para clasificaciones linealmente separables, dando lugar a resoluciones de carácter logístico. Esto significa que los valores de salida asociados con las clases, que para el caso del perceptrón simple son dos, se diferencian bastante (dos ejemplos: (-1 o 1), (0 o 1)), por lo que no suponía un problema complejo.

El perceptrón simple, como unidad neuronal, necesita de un umbral para la clasificación de los valores de entrada en categorías o clases, según lo obtenido a partir de las operaciones del proceso. Para ello se hace uso de una función de coste y de un algoritmo de aprendizaje que permiten realizar la regresión logística, que sirve para mejorar esta clasificación por clases. La función de coste permite determinar el error de predicción del modelo entre la hipótesis o las operaciones del proceso y el resultado esperado. El algoritmo de aprendizaje es la segunda operación, consigue recalcular los pesos o parámetros empleados en las operaciones del proceso y su cálculo implica a los pesos, los valores de entrada y la tasa de aprendizaje.

El perceptrón, representado en la **Figura 2.1**, recibe valores binarios de entrada a los cuales se aplica un peso que da cierta preponderancia a cada valor, después realiza su suma, aplica una función de activación de tipo escalón (pone a ‘ON’ u ‘OFF’ a cada uno los valores de entrada) y devuelve los valores de naturaleza binaria a la salida, dependiendo del umbral para realizar su clasificación. Mediante un proceso de aprendizaje iterativo y la regla de

aprendizaje derivada de la ley de Hebb[2] se consigue ir clasificando los valores en sus respectivas clases modificando los pesos iniciales, corrigiendo los errores de las primeras clasificaciones hasta terminar de delimitar las clases, haciendo uso de un factor de aprendizaje.

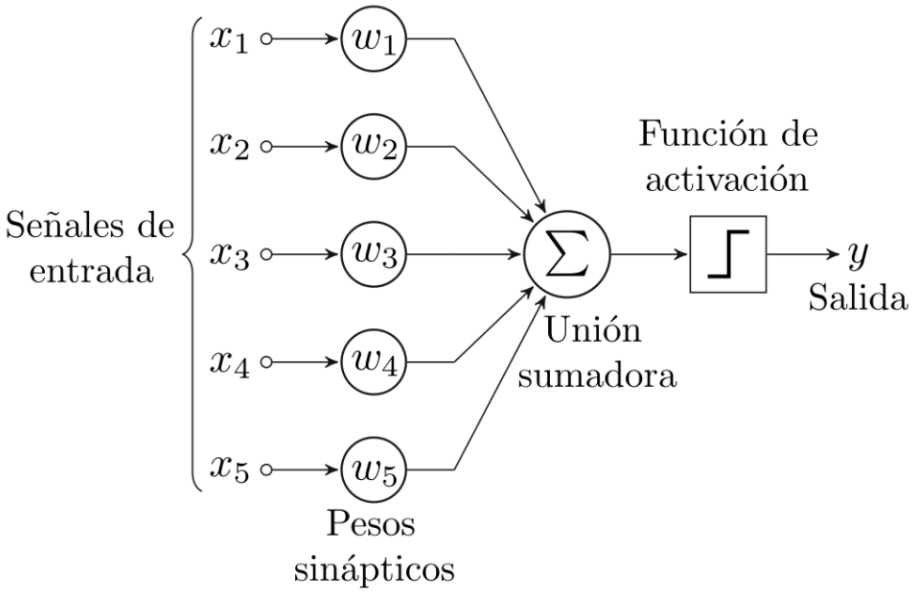

Figura 2.1: *Representación del perceptrón. Figura extraída de la nota* [3]

Debido a que los problemas reales son más complejos, se empezó a introducir más unidades neuronales formando capas que conectaban cada entrada con capas intermedias u ocultas formadas por otras unidades neuronales y estas a su vez a otras capas ocultas o a la capa de salida, dando lugar a las denominadas redes neuronales profundas y al aprendizaje profundo (Deep Learning). También se consiguió que este resultado fuese más allá de los simples estados binarios, introduciendo operaciones de optimización. Así, se desarrollaron problemas sofisticados donde las clases no fuesen linealmente diferenciables de soluciones más complejas. Esto se tradujo en las RNA de perceptrones multicapa, añadiendo capas

[2] https://psicologiaymente.com/neurociencias/ley-de-hebb

[3] https://es.wikipedia.org/wiki/Perceptr%C3%B3n

ocultas a la red la cual se formaba anteriormente por un perceptrón simple, que influyeron en casi todos los desarrollos posteriores de RNA.

Debido a que ya no se podían separar linealmente las clases, se sustituyó la función escalón por la función de activación logística sigmoidea, que conseguía transformar las neuronas binarias en sigmoideas**,** modificando la función de coste. Esta función se representa en la **Figura 2.2**. Además, se implementó el algoritmo de aprendizaje con retropropagación (*backpropagation*) para mejorar los resultados finales. Esto será descrito en el apartado 2.2.1.

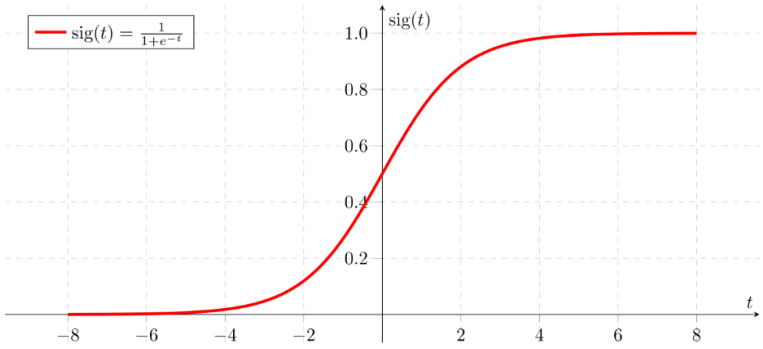

Figura 2.2: *Función de regresión logística sigmoidal con frontera de decisión 0.5. Figura extraída de la nota* [4]

Nuevas investigaciones consiguieron implementar diferentes tipos de RNA, como es el caso de la red neuronal convolucional (CNN, *Convolutional Neural Network*), aplicada sobre todo para imágenes como datos de entrada, en la cual la neurona funciona de manera totalmente distinta. Más adelante, en el apartado 2.2.2 será detallado.

Para este tipo de red como para otro tipo de redes, el número de capas ocultas tendía a ser alto. Esto conllevó a enfrentar problemas totalmente distintos donde la complejidad era mayor y que podían causar el sobreajuste de los parámetros (*overfitting*), es decir, el sobreajuste de la red al ajustarse en exceso a algunos valores de entrada de determinada clase, dificultando predicciones ajustadas para otras clases. Para resolver esto, se pensaron diferentes métodos de resolución a aplicar, métodos como el de aplicar en los programas de

[4] https://de.wikipedia.org/wiki/Sigmoidfunktion

las RNA bucles de numerosas interacciones o episodios, ajuste fino (*fine-tuning*, que consiste en añadir capas a la red conectadas a la última capa), etc.

Finalmente, las redes neuronales hacen uso de métricas de distancia tales como las distancias euclídeas, coseno. *Mahalanobis*, etc. Estas métricas se aplican en diferentes métodos, para la aplicación del cálculo de distancias en espacios multidimensionales, con el objetivo de establecer la similitud entre dos ejemplos [3]. El uso de estas métricas ayuda a relacionar objetos o ejemplos de formas similares para organizarlos en un mismo grupo o clase.

2.2.1 Retropropagación

El algoritmo de retropropagación, tiene por finalidad extraer las interpretaciones del error de predicción en cada nodo de la red para minimizar la función de coste y obtener mejores resultados [4].

Consiste en la aplicación de diversas operaciones. Funciones como el estimador del error de la función de coste y un optimizador de la función de coste, que permiten utilizar como algoritmo de aprendizaje el descenso por gradiente: consiste en recalcular los pesos o los parámetros de las neuronas del modelo paramétrico y el uso de la función de coste y la derivada para su cálculo. Esto realiza la búsqueda de un camino para el descenso por el cual se pueden encontrar los mínimos globales, aunque puede darse la posibilidad de complicarse su búsqueda cuando hay varios mínimos locales para una misma interacción. Estos mínimos locales se pueden analizar mediante una gráfica donde se comparan las pérdidas y los pesos, pudiendo verse en 2D o en 3D según el caso.

El cálculo del error puede realizarse mediante diversos métodos como el estimador de error cuadrático medio (ECM), la raíz del error cuadrático medio (RECM), el error medio aritmético (EMA), la entropía cruzada categórica, etc.

La optimización puede realizarse mediante el método del descenso estocástico del gradiente (SGD, *Stochastic Gradient Descent*), el método Adam, Adagrad, etc.

2.2.2 Red Neuronal Convolucional

Una red neuronal convolucional (CNN, *Convolutional Neural Network*) es una RNA fruto del desarrollo del perceptrón multicapa, la cual hace uso de matrices bidimensionales [5].

Consiste en una red que asocia cada neurona a un píxel o grupo de píxeles. El tamaño de las imágenes de entrada resulta determinante, ya que no es óptimo para imágenes de alta resolución el caso de un solo píxel por neurona. Aplicar grupos de píxeles por cada neurona hace que la red pueda analizar un trozo de una imagen por cada neurona y al asociar cada trozo con estados próximos consigue reconocer trazos característicos y así hasta formar los mapas de características (*feature maps*) de una imagen completa, permitiendo asociar imágenes procedentes de un conjunto predeterminado, de un *dataset*, en clases al concluir los pasos de la red.

Para cada fase o capa, la red aplica un tipo de operación distinto a los datos que de entrada tendrían forma de matriz para el caso de una imagen de tonalidades grises, como se representa en la **Figura 2.3**, que describe el funcionamiento de una red CNN. La red empieza a funcionar aplicando un bloque de convolución, que genera los primeros mapas de características. Esta función se aplica repetidas veces, logrando en cada aplicación reconocer mejor los trazos de la imagen. El mapa de características consiste en los datos modificados a partir de las operaciones de la red sobre las imágenes de entrada, que determinan las particularidades de cada imagen y que finalmente se verán transformados en los resultados relacionales, los cuales son extraídos del módulo definido en el apartado 4.2.2.

El bloque convolucional aplica en primer lugar la función de convolución, pero no una sino múltiples convoluciones, como se puede observar en la imagen, ya que estas convoluciones consisten en aplicar numerosas máscaras que forman un banco de filtros para conseguir que la matriz de los datos de entrada se transforme en un determinado número de matrices de datos modificados a su salida. Esto se logra en cada convolución mediante una máscara (*kernel*), que aplica el producto escalar a cada pedazo de la imagen o neurona y logra que grupos de píxeles o neuronas (datos de la matriz) se transformen en un solo dato generando una nueva matriz pequeña de nuevos datos. Finalmente, el bloque convolucional aplica una función de activación. Una vez llegado a este paso se obtendrían los primeros mapas.

Después se aplica la operación de submuestreo, ya que no se busca manejar grandes volúmenes de datos, la red CNN tiene como objetivo economizar el uso de la memoria y tiempo invertido, aplicando una capa de agrupación o sobremuestreo (*subsampling*) para agrupar datos cuando son muy extensos y finaliza con la capa de alta conectividad (*full connection*), convirtiendo las matrices en un vector que combina las todos los mapas

obtenidos y aplicando operaciones que transforman la linealidad y el rango de los datos de salida.

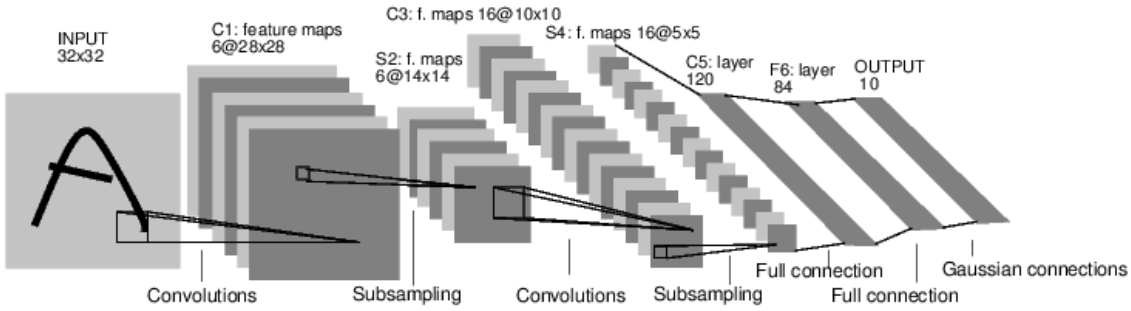

Figura 2.3: *Ejemplo de CNN, Cifar10. Figura extraída de [6]*

Una vez terminado este proceso, la red obtiene el resultado relacional (*relation scores*) a los cuales se podría dar una finalidad u otra. Para el programa propuesto por el artículo de referencia se aconseja el uso en dos fases: una fase de entrenamiento y otra de prueba.

3 Diseño

3.1 Introducción

En esta sección de definen las características del programa objeto del Trabajo de Fin de Grado.

3.2 Definición del programa a realizar

La red neuronal de este proyecto consiste en una red relacional (RN, *Relation Network*), que proporcionará la capacidad de extraer relaciones entre objetos. Está basada en el tipo de RNA CNN, como queda comentado en la sección del *Estado del Arte*, que introduce los datos desde la entrada de la red para almacenar los mapas de características (extraídos y modificados en cada etapa) a la salida (*feed-forward*).

La red se ha diseñado para realizar un aprendizaje profundo que, mediante retropropagación y las operaciones implicadas en la práctica, conseguiendo extraer unas pérdidas lo más ajustadas para un entrenamiento óptimo a partir de los mapas extraídos.

Está compuesta por dos ramas con funcionalidades distintas, como se puede ver en la **Figura 3.1**, siendo $f\varphi$ y $g\varphi$ las operaciones de cada módulo que generarán los mapas: en el primer módulo se busca extraer los mapa de características intermedias, en el segundo inicialmente se realizan las concatenaciones de los mapas entre los distintos tipos de imágenes y finalmente las operaciones para obtener los resultados relacionales, es decir, los resultados buscados.

Para ello, se ha desarrollado una aplicación en el lenguaje de programación *Pytorch* de *Python* que tiene en cuenta dos modelos de reconocimiento por clases con pocas imágenes, para aprender y clasificar.

Primero se ha desarrollado el programa que realiza el modelo de una imagen de muestra (1-shot) por cada clase y los pesos a aplicar, para luego poder generalizarlo y conseguir el modelo de reconocimiento para pocas imágenes de muestra (*few-shot*). Posteriormente, en la fase de pruebas se han usado cinco imágenes por muestra (5-shot) para realizar las comprobaciones.

Además, el programa está configurado para clasificar imágenes en cinco clases distintas, consiguiendo un modelo de cinco vías (5-*way*).

El artículo de referencia también desarrolla un modelo de cero imágenes de muestra (*zero-shot*) que en este proyecto no se ha implementado.

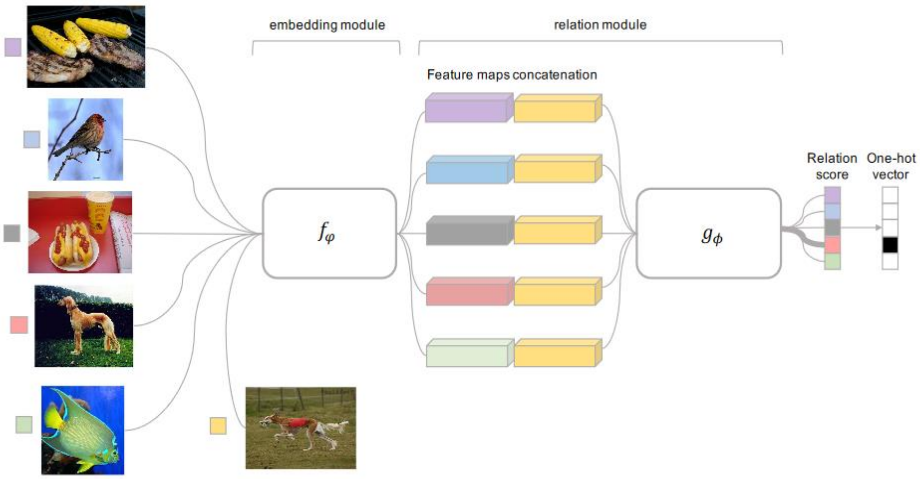

Figura 3.1: *Estructura del programa para 5-way 1-shot. Figura extraída de [1]*

Por último, para una comprobación exhaustiva, se han realizado diversas pruebas para una y cinco imágenes de muestra con un número C de clases, para lograr comparar imágenes de muestra (*sample*) con las imágenes de consulta (*query*), **Figura 3.2**, siendo x̂ las imágenes del *dataset* e ŷ las clases del *dataset*, con la finalidad de realizar las predicciones para las distintas clases que contenga el *dataset* escogidas de forma aleatoria. Según el modelo, habrá por cada clase K imágenes de muestra, que afectará al número de imágenes y a las operaciones implicadas. Las imágenes de consulta son siempre fijas. Para este caso habrá 15 imágenes de consulta. La morfología de la imagen o de las imágenes de muestra determinará la buena o mala predicción final del programa.

$$\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{m} \ (m = K \times C)$$
$$\mathcal{Q} = \{(x_j, y_j)\}_{j=1}^{n}$$

Figura 3.2: *Set de imágenes muestra y consulta. Figuras extraídas de [1]*

3.3 Dataset

Para probar el código se ha utilizado el *dataset* de *miniImageNet* para realizar el entrenamiento. Este *dataset* cosiste en una base de 60.000 imágenes a color 2D, con formato PNG, de tamaño 84x84 píxeles, agrupadas por carpetas que corresponden cada carpeta a una clase, habiendo en total 100 clases distintas de imágenes.

Para su uso, se ha de realizar una transformación para obtener un rango distinto al original, pero adecuado, de valores normalizados de las imágenes en el rango [-1,1], que será óptimo a la hora de aplicar las funciones.

Una vez entrenada la red neuronal, se ejecuta la fase de prueba o test con este mismo *dataset*, pero modificado, con el fin de ajustarle a un volumen de imágenes y clases adecuado para realizar la prueba con pocas imágenes.

Además, se han elaborado *datasets* propios compuestos de un número igual de imágenes y de clases. Todos estos *datasets* están formados por 20 imágenes por clase, en un total de 5 clases para ver el funcionamiento de la red con *datasets* pequeños. Para crear estos *datasets*, se han usado herramientas facilitadas en internet de cambio de formato y de redimensión de imágenes para tener el tamaño anteriormente indicado.

Con el código realizado, se entrena la red neuronal para que pueda clasificar imágenes cogidas aleatoriamente de cada clase, siguiendo el esquema ya mencionado, y según las funciones aplicadas extraerán unos mapas de características que determinarán la clasificación con cada clase, y salvo un margen de error obtenido (grande, mediano o pequeño), las clasifica correctamente.

3.4 Herramientas de Programación

Python es el lenguaje de programación en el que se ha generado el código, mediante los paquetes *Pytorch* (*torch*, *torchvision*, etc.), *Matplotlib*, *Numpy*, etc., instalados en la terminal

de Linux de una máquina virtual. Pytorch es usado para el procesamiento de tensores debido al uso de una gran cantidad de datos procedentes de imágenes y para conseguir almacenar decenas, cientos o miles de matrices en un mismo tensor. Los programas que necesiten procesar tensores en gran cantidad tardarán en realizarse mediante el uso de las unidades centrales de procesamiento (CPU en inglés), que hasta hace poco eran el elemento fundamental de cualquier ordenador.

En 2007, Nvidia desarrolló las unidades de procesamiento rápido (GPU en inglés) permitiendo abordar de mejor manera aquellos programas con un gran volumen de datos a procesar, logrando facilitar el desarrollo de programas informáticos acerca de IA. Su gran capacidad ha conseguido sustituir a las CPUs por completo en algunas funcionalidades de ordenadores que necesitan un alto rendimiento.

La diferencia principal entre estos dos tipos de unidades de procesamiento es la cantidad de núcleos que tiene cada uno. Estamos hablando de su hardware, que para el CPU tenemos una arquitectura de varios núcleos óptimos (hoy en día han alcanzado hasta 16 núcleos) en serie secuencial mientras que para la GPU podemos tener hasta miles de núcleos haciendo funcionar procesos en paralelo, pero de tamaño más pequeño y más eficientes. Esto supone una capacidad mayor de procesamiento de código, usado en buena parte de las aplicaciones destinadas a la inteligencia artificial o a aplicaciones que necesiten una gran velocidad de procesamiento. Para muchas aplicaciones, un uso compartido de las memorias GPU y CPU consigue optimizarlas, ya que no necesitan del alto rendimiento GPU salvo en una pequeña parte del proceso [7].

En el procesado mediante GPU del programa, la fase de entrenamiento realizado por el tutor del TFG, se ha necesitado la instalación de los paquetes de CUDA (Compute Unified Device Architecture) que es una plataforma de cómputo paralelo que usa los núcleos de GPU para procesar datos de programas distintos a la vez, es decir, en paralelo, usando el lenguaje en C/C++ de programación y pudiéndose aplicar Python, mediante wrappers [8]. Para que esto sea posible, se necesita tener una tarjeta Nvidia GPU que permita usar CUDA. Y, por lo tanto, tendremos que instalar el CUDA Toolkit y también Conda desde la web de Pytorch [9], que es un gestor de paquetes que ayuda en la instalación de CUDA.

4 Desarrollo

4.1 Introducción

En esta sección se describe el desarrollo teórico del programa realizado en *Pytorch*. Los primeros pasos para realizar este programa se han dado usando el modelo ofrecido por los desarrolladores de *Pytorch* [6] que se encuentra en su página web, referenciada en [9]. Este modelo se ha usado como guía y sirve precisamente para desarrollar las redes neuronales mediante sus módulos *torch.nn.*

4.2 Desarrollo de la Red Neuronal

La red neuronal de carácter RN, representada en la **Figura 4.1**, se divide en dos módulos: de incrustación (*embedding module*) y relacional (*relation module*). Está basada en la consecución de bloques convolucionales, en la operación de agrupación y en las diversas operaciones que se comentarán con posterioridad.

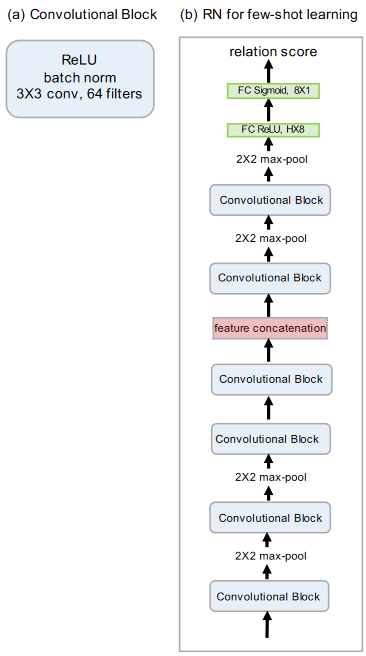

Figura 4.1: *Estructura de toda la red neuronal y el bloque convolucional. Figura extraída de [1]*

El bloque convolucional (*convolutional block*) se compone de las siguientes operaciones 2D: una convolución con máscaras 3x3 que forman un banco de 64 filtros, normalización por lotes y una operación no lineal *ReLU*.

Con esto se consigue transformar el conjunto de datos procedentes del conjunto de imágenes seleccionadas aleatoriamente, según cinco vías y K número de imágenes (5-*way* K-*shot*), haciendo que los datos sean más pequeños y obteniendo los mapas de características del proceso. Esto se puede observar en la **Figura 4.2.**

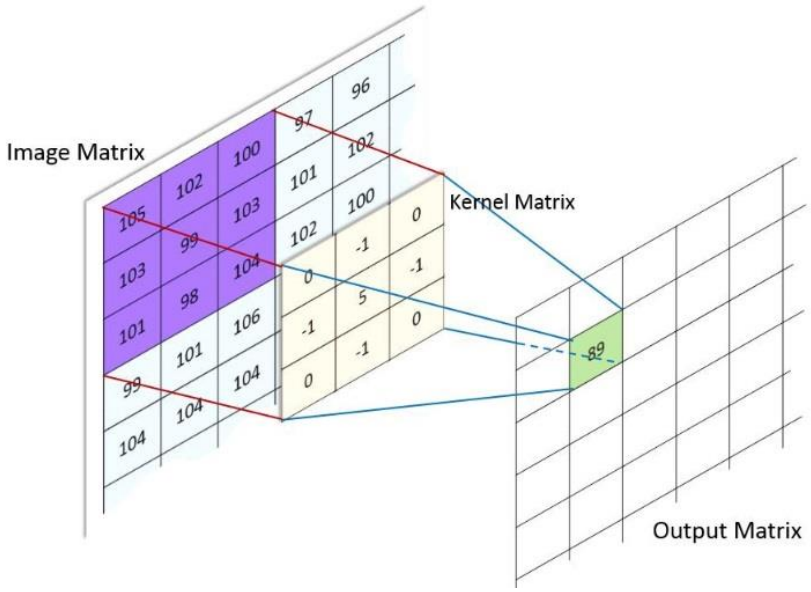

Figura 4.2: *Ejemplo de obtención de los mapas de características mediante la operación de Convolución. Figura extraída de la nota* [5]

La función de convolución, explicada parcialmente en la sección *Estado del Arte*, contiene 64 máscaras, cada una de tamaño 3x3 aplicadas cada vez sobre los píxeles de cada imagen sin relleno de ceros (*padding*), por lo tanto, no añade ceros alrededor de la imagen y como resultado obtiene una matriz de tamaño menor que la original, ya que no aplica la máscara de igual manera para cada uno de los píxeles.

En primera instancia, se obtienen por entrada imágenes a color que se guardan en un tensor, por lo tanto, en vez de aplicar cada máscara a un solo componente de tonalidad gris, se aplica

[5] https://medium.com/analytics-vidhya/layers-of-a-convolutional-neural-network-168daddd2dd1

cada máscara sobre los tres componentes de color por separado y luego se suman todos los resultados aplicando el sesgo inicial. El sesgo inicial será explicado después.

Una vez aplicadas todas las máscaras habrá un total de 64 componentes o matrices. Con la normalización por lotes de cada mini lote o grupos de valores de cada entrada, se consigue reducir el tiempo de entrenamiento y permite mayores tasas de aprendizaje, debido a la reducción de la covarianza por desplazamiento interno. Este desplazamiento modifica la distribución de las entradas a las capas siguientes en vez de permanecer continuas.

La operación de activación *ReLU* hace que los valores a la entrada pasen de estar en todo el dominio de los números reales a tener solo valores positivos y ceros y por consiguiente activa o desactiva las capas según el resultado sea cero o distinto de cero. Esto es similar a los estados 'ON' y 'OFF' lógicos, pero en este caso son no binarios que consiguen nuevos valores de entrada que estabilizan al sistema [10].

Acto seguido, la operación de agrupación 2x2 (2x2*max-pool*) agrupa los valores de cada componente del tensor para disminuir el tamaño de los mapas de características consiguiendo trabajar más eficientemente y sin tanta carga computacional. Para ello realiza una media de los valores mediante bloques de tamaño 2x2 píxeles, consiguiendo disminuir el tamaño de cada matriz.

El tensor antes mencionado, que se compone de una o varias imágenes de entrada según el modelo de una o cinco imágenes de muestra (K-*shot*), pasa de tener una matriz por cada color para cada una de las imágenes de entrada de muchos datos y grandes a tener una matriz por cada filtro aplicado en cada operación de convolución a cada imagen, consiguiendo un tensor de menos datos y pequeños debido a las operaciones aplicadas.

Después de la aplicación sucesiva de estas operaciones y otras operaciones que se detallarán en los siguientes apartados, se obtiene a la salida de la RN el resultado relacional formado por un vector de datos.

Finalmente, para esta RN, se ha precisado de un cálculo de distancias mediante un método de aprendizaje, tanto de incrustación como de métricas no lineales profundas, evitando escoger una función distancia permitiendo identificar mejor las relaciones de acierto [1].

4.2.1 netEncoder

Esta parte de la red consiste en desarrollar el módulo de incrustación (*embedding module*), en los primeros pasos, como se indicó en la **Figura 4.1**.

Este módulo contiene cuatro bloques convolucionales como los explicados anteriormente, pero de parámetros distintos, y es debido a que, a la entrada del *netEncoder*, se recibe el conjunto de imágenes 2D procedentes del *dataset*, tanto de muestra como de consulta, almacenados en el tensor con dimensión diferente a la dimensión que se obtiene a la salida de cada interacción del bloque convolucional, que hace necesario el ajuste de los valores de cada bloque.

Corresponde, para cada imagen de entrada, dimensiones de los tres componentes de color y con la primera interacción del bloque se transforma y se aumenta a 64 componentes debido a los filtros del bloque, como quedó explicado anteriormente. Una vez realizada la convolución sobre la entrada, se aplica la normalización y la función de activación *ReLU*, consiguiendo evitar el cambio de sesgo que es común al aplicarse la función *ReLU*, ya que sólo la función *ReLU* no centraría los valores entorno a cero, aumentando el sesgo y, por lo tanto, siendo más difícil llegar a tener valor 1 a la salida, requiriendo valores de entrada más altos. El sesgo resulta de la diferencia del estimador (función de coste) y el verdadero valor del parámetro a estimar.

Después, se agrupan en las dos primeras interacciones de los bloques los puntos de cada capa, disminuyendo sus tamaños para eliminar datos irrelevantes, es decir, submuestreando. Así, se podrá operar con los datos sin utilizar tanta memoria de procesamiento y agilizar el programa.

Finalmente, a la salida del *netEncoder*, nos queda un mapa de características pertenecientes a las imágenes de muestra y de consulta, que sirven para generar el tensor de concatenaciones entre los mapas de las imágenes de muestra y las imágenes de consulta, que son usadas en la siguiente parte de la RN.

4.2.2 netRelation

Esta parte de la red consiste en el desarrollo de la parte del módulo relacional (*relation module*), indicado en la **Figura 4.1**, que realiza la comparación de los mapas extraídos del

módulo de incrustación *netEncoder* y concatenación de las imágenes de los dos tipos: de muestra y de consulta.

Para cada modelo, de una o cinco imágenes de muestra, se procura obtener unos valores de entrada, conformados por las concatenaciones de mapas extraídos de las distintas agrupaciones de K imágenes de muestra con los mapas de cada imagen de consulta, guardadas en cada posición de un tensor concatenación.

La diferencia entre los modelos está cuando se crean las concatenaciones entre imágenes de muestra con las de consulta. Con el modelo de una imagen de muestra se concatena por clase cada imagen de muestra de forma individual con cada imagen de consulta, mientras que con cinco imágenes de muestra se suman los mapas de las cinco imágenes de muestra de cada clase, concatenándose con cada imagen de consulta. Esto último consigue que haya más datos que faciliten el reconocimiento de una imagen para su clase, para cada imagen de consulta.

La concatenación prepara la nueva parte de la red para sacar el nivel de relación de las imágenes de consulta con las clases. Entonces, tenemos *netRelation* que comienza como el anterior bloque, pero con distintos parámetros con respecto al número de matrices con las que se operan en los bloques convolucionales. Aplica una función de reorganización de *arrays*, dos transformaciones lineales y las funciones de activación *ReLU* y Sigmoidea.

Todo esto reorganiza las columnas y filas del tensor, aplica las dos transformaciones lineales con distintos parámetros y a cada una se le aplica una transformación no lineal: a la primera se le aplica la función *ReLU* y a la segunda la función logística sigmoidal. Esta última función, como se indicó en la sección *Estado del Arte*, consigue extraer unos valores pequeños, en un rango estrecho y, gracias a la función *ReLU* que desprecia los valores negativos, se obtiene como resultado final de la red el resultado relacional. Permite obtener los pesos del entrenamiento o los resultados de porcentaje de acierto en el test.

El resultado que se obtiene de aplicar la *netRelation* es el vector formado por el resultado relacional. Resumidas las operaciones implicadas en **Figura 4.3**, son $f\varphi$ los mapas de características del módulo de incrustación fila y columna, $C(\cdot,\cdot)$ es la operación de concatenación que compara esos mapas y $g\varphi$ el módulo relacional.

$$r_{i,j} = g_\phi(\mathcal{C}(f_\varphi(x_i), f_\varphi(x_j))), \quad i = 1, 2, \ldots, C$$

Figura 4.3: *Resultados relacionales para cada imagen consulta de entrada. Figura extraída de [1]*

El vector del resultado relacional que se forma por los mapas de características extraídos de todas las interacciones posibles del programa, en primera instancia está compuesto por una única columna y se amplía hasta cinco columnas. Ello supone desplegar y organizar los valores obtenidos según el número de clases mediante una función que permite aplicar las operaciones finales pertinentes del entrenamiento y de las pruebas.

Los valores del vector resultado son de magnitud pequeña, en un rango determinado debido a la función sigmoidal e indicando para cada imagen de consulta o fila a qué clase o columna se han relacionado. El valor perteneciente a las columnas del vector que sea de mayor valor indica la clase predicha ya que expresa mayor coincidencia.

4.3 Cuerpo

Como apunte previo a definir el entrenamiento y el test, estas dos pruebas hacen uso de los dos módulos de RN, la manera de formar los tensores de las clases, de los tensores de imágenes formadas por imágenes de muestra y de consulta y el proceso de concatenación. La finalidad del resultado relacional en cada una es completamente distinta.

4.3.1 Fase de Entrenamiento

El entrenamiento para pocas imágenes está basado en el meta-aprendizaje, ideal para que el programa aprenda a aprender y de manera rápida a partir de muchas imágenes, con modelos y habilidades distintas o adaptarse a nuevas especificaciones de manera que no cueste gran esfuerzo, logrando que los parámetros o pesos que se necesitan en el test, se recalculen en base a un bucle que tiene por objetivo optimizarlos [11], y el enfoque a seguir trata de medir las métricas entre diversas imágenes para compararlas y que, a diferencia de otros enfoques, no se basa en redes neuronales relacionales (RNN, *Relational Neural Network*), que usa memorias e implementa realimentación en los nodos, ni realiza ajuste fino. Ejemplos de RNN los veremos comparados con los resultados de este proyecto en el apartado 5.3.

En el entrenamiento, se necesitan realizar diversas mediciones para tener una buena base y asegurar el correcto funcionamiento del programa, generando multitud de episodios con todas las operaciones involucradas en el proceso y guardando los pesos adquiridos para aplicarlos en la fase de pruebas para la consecución de los mejores resultados.

En el entrenamiento, se aplican pesos iniciales, ayudándonos por sesgos iniciales (bias), que agregan a la red neuronas que se conectan con diversas capas, ofreciendo a las redes unos parámetros útiles a la hora de poner en marcha el programa, como inicialización de las capas, siempre y cuando se evite que provoquen saturación. También se establece un número total de 100.000 episodios, tasa de aprendizaje en 10^-3 y se aplica la función ECM para realizar la retropropagación, encargada de convertir el entrenamiento en un entrenamiento extremo a extremo. La operación ECM, fórmula representada en la **Figura 4.4** (siendo el primer vector el de las predicciones realizadas y el según vector como el de los valores óptimos), utiliza el vector del resultado relacional extraído de *netRelation* con los valores reorganizados explicado anteriormente.

$$\mathrm{ECM} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

Figura 4.4: *Fórmula ECM*. Figura extraída de la nota [6]

Todo esto contribuye a calcular los errores en retropropagación, como se ha visto en el *Estado del Arte*. Para hallar unos valores óptimos de los pesos, usando el cálculo del error por cada capa, la retropropagación comienza con los valores obtenidos por la función ECM. Este estimador del error, visto en la **Figura 4.5**, frente a otras alternativas, es continua y genera una pendiente que decrece cerca del mínimo. También tiene en cuenta la tasa de aprendizaje para un aprendizaje por mini lotes, que sirve para conseguir un aprendizaje rápido, cuidando de no generar fluctuaciones aleatorias provenientes de los gradientes implicados en el descenso por gradiente en caso de ser alta la tasa o, por el contrario, hace disminuir la velocidad de aprendizaje teniendo que ajustar la tasa si es muy pequeña. Mientras tanto, se disminuye la tasa cada cierto número de episodios.

[6] https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio

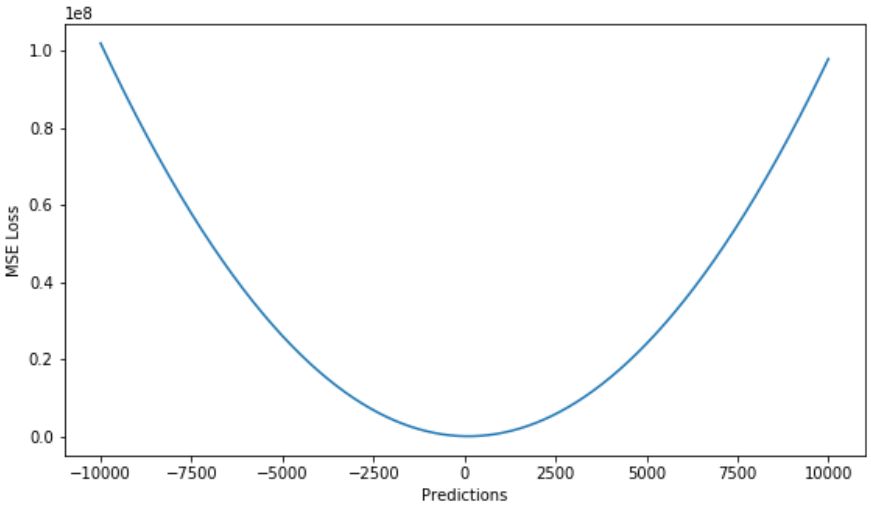

Figura 4.5: *Representación de la función ECM*. Figura extraída de la nota [7]

Para obtener unos parámetros apropiados, es necesario el uso de los algoritmos de optimización. El empleado en este proyecto es el de Adam [12], que mejora a SGD indicando cuánto tiene que avanzar en la dirección del descenso por gradiente al ajustar la tasa de aprendizaje según el estado de dispersión de los parámetros. Esto finalmente corrige de la mejor manera posible los pesos de la red disminuyendo los errores o pérdidas, ya que, por cada capa de la red en retropropagación, se recalcula la función de coste con esos parámetros de mejora de la predicción del error por cada episodio. Todo esto tiene por finalidad el procurar que ocurra un descenso adecuado. Así puede encontrar los mínimos locales óptimos de los parámetros de nuestra función en cada capa de la red por cada mini lote.

Hay que señalar también que esta solución no está exenta de problemas. Por eso, se busca que converja de la mejor manera posible mediante descenso por gradiente, visto de forma sencilla en la **Figura 4.6**, donde $E(\omega)$ es la función de coste por cada parámetro ω, η es la tasa de aprendizaje y ω_{min} es el mínimo global al cual es deseable converger para hallar la mejor solución.

[7] https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

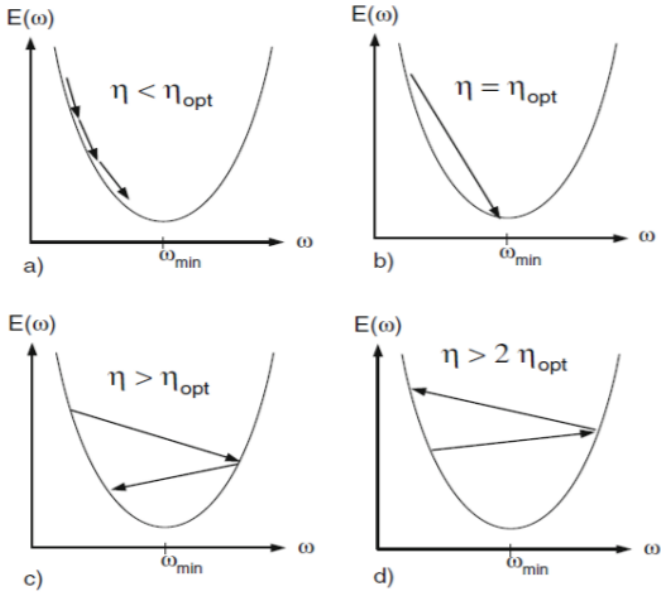

Figura 4.6: *Ejemplos de funcionamiento de la optimización del descenso por gradiente en 2D, para tasas de aprendizaje de menor valor a) hasta de mayor valor d) con respecto a una tasa óptima. Fuente extraída de la nota* [8]

Una forma de ver el problema a resolver es en la representación en 3D de la **Figura 4.7**. Se visualiza el mínimo global, local y el punto de silla, como parte del problema que ha de solucionar el descenso por gradiente. Pero es difícil hallar una tasa de aprendizaje que haga converger de forma perfecta al programa, por lo que nos bastará con que logre unos parámetros deseables y cercanos al mínimo.

Se concluye que, para cada salida de nuestra red, hay un proceso en cadena hacia atrás que comprueba qué entradas de capas ocultas o anteriores afectan a cada nodo o salida y en qué medida el manejo de las diversas operaciones conducirá a recalcular los pesos para cada nodo. El objetivo final es realizar un aprendizaje óptimo. Los pesos conseguidos son guardados en un archivo que es usado en la fase de prueba.

[8] https://elvex.ugr.es/decsai/deep-learning/slides/NN6%20Optimization.pdf

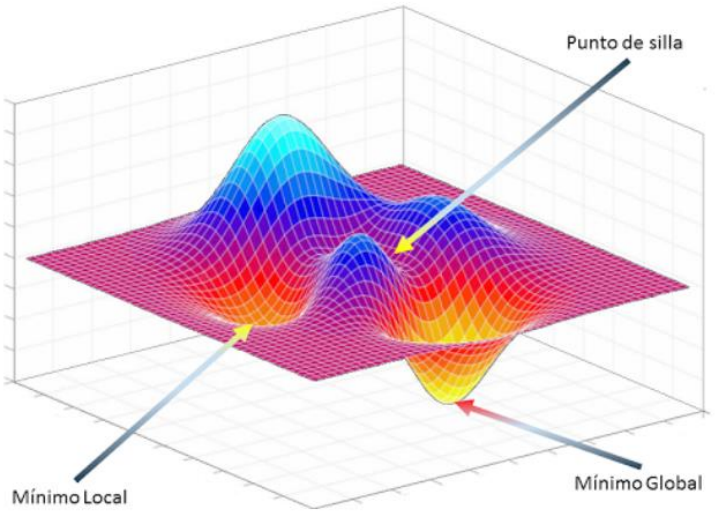

Figura 4.7: *Visualización en 3D del problema a resolver mediante el algoritmo de descenso por gradiente. Figura extraída de la nota* [9]

4.3.2 Fase de Pruebas

En la fase de pruebas, se han utilizado los pesos generados del entrenamiento de los numerosos episodios realizados para ponerlos en práctica iniciando las redes. Esto hace que no sean necesarios tantos episodios como los invertidos en la fase de entrenamiento, ya que para la fase de pruebas se usan pesos ajustados que inicializan cada uno de los módulos de la RN del test.

La fase de pruebas hace uso del resultado relacional y del tensor de clase para comparar los resultados de las imágenes de muestra y de consulta. Con un número total de 600 episodios se obtienen valores de predicción aceptable.

Los resultados obtenidos en estas pruebas se describen en el capítulo siguiente.

[9] http://numerentur.org/gradiente-descendente/

5 Pruebas y resultados

5.1 Introducción a las pruebas y a los datasets utilizados

En esta sección se mostrará si el programa desarrollado, basado en el artículo de referencia, proporciona resultados óptimos en el proceso de reconocimiento de imágenes.

Para ello, en la fase de entrenamiento, se ha usado el *dataset* de *miniImageNet* con un número alto de episodios, para generar los pesos y luego realizar las pruebas con distintos *datasets* de menor tamaño y con menor número de episodios.

Por lo tanto, para cada episodio se ha formado un tensor de imágenes aleatorias que interactúa con las imágenes de consulta en el momento de la concatenación. Las imágenes de consulta, de 15 imágenes aleatorias por cada clase, son las imágenes a clasificar en su clase o en otra de forma accidental. Cada imagen de muestra será extraída aleatoriamente de una de las clases que forman cada *dataset*.

Para ver si los resultados extraídos por el programa son óptimos, se han comparado con los indicados en el artículo de referencia y con los obtenidos por otros investigadores que realizaron otros modelos diferentes.

El juego de *datasets* utilizados en las pruebas son los siguientes:

- *miniImageNet:* Este conjunto de imágenes se ha modificado en este trabajo para formar dos *datasets* diferentes para pocas imágenes, eliminando clases sobrantes e imágenes por clase hasta tener 5 clases y 20 imágenes por cada clase.
    - En el primero de ellos hay varias clases similares, que sirven para ver el comportamiento de nuestro programa ante un caso como este, que será en el *dataset miniImagenet* 1: cangrejo, microorganismo, perro 1, perro 2 y perro 3.
    - El segundo, *miniImagenet* 2, está formado por: cangrejo, microorganismo, perro, guitarra y reloj de arena.
- EPS: Está formado por las clases servicio, aula, laboratorio, comedor y sala.

- Edificios UAM: en este *dataset están* las clases EPS, monumento, plaza mayor, rectorado y posgrado.

Este conjunto de *datasets* ofrece una idea general del funcionamiento del programa desarrollado en este TFG. En la **Tabla 5.1** se muestran los resultados obtenidos por el programa para cada uno de los *datasets* anteriores.

Datasets	**5-way Acc.**	
	1-shot	5-shot
miniImageNet 1	44.54 ± 0.23%	59.54 ± 0.49%
miniImageNet 2	50.90 ± 0.21%	66.19 ± 0.13%
EPS	53.62 ± 0.25%	72.08 ± 0.21%
Edificios UAM	**58.46 ± 0.32%**	**78.03 ± 0.21%**

Tabla 5.1: *Resultados de la RN mediante diversos datasets, remarcados en negrita los mejores casos, para 600 episodios de la fase de pruebas.*

5.2 Pruebas con Datasets

En esta sección se han analizado los resultados obtenidos por el programa para los dos *datasets* que han proporcionado los valores opuestos en reconocimiento de imágenes de acuerdo a los resultados indicados en la tabla 5.1:

- *miniImagenet 1: Dataset* con los peores resultados.
- *Edificios UAM*: *Dataset* con los mejores resultados.

5.2.1 Resultados de las pruebas para el dataset "minilmagenet 1"

En este apartado, se realizan las comprobaciones del modelo de una imagen de muestra para el *dataset* "*miniImagenet* 1", siendo este *dataset* el de menor porcentaje de acierto.

En el último de los episodios del programa, se obtuvo un porcentaje acumulado de acierto de 44.54%, donde la primera imagen de muestra es de la clase cangrejo, la de mayor acierto y las imágenes de consulta de su misma clase, son las de la **Figura 5.1**.

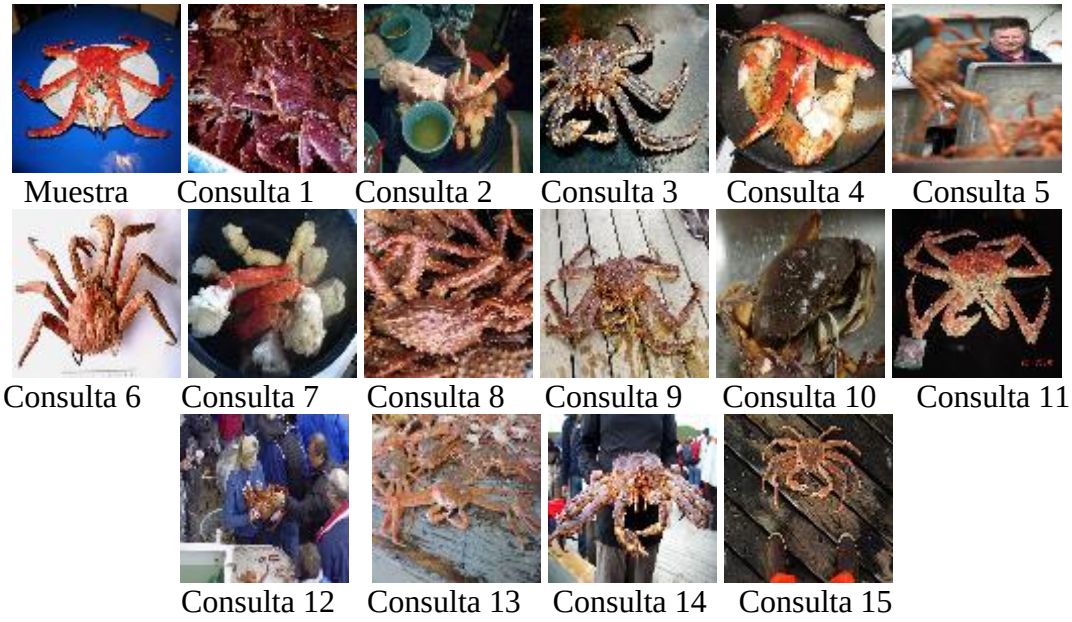

Figura 5.1: *Interacción de una imagen de muestra con las 15 imágenes de consulta de la clase cangrejo.*

Los aciertos obtenidos para esta clase son las Consultas 1, 3, 5, 6, 9, 11, 12, 13, 14 y 15. Se observa que, de las 15 imágenes de consulta de su propia clase, acierta 10. Parece que el diseño consigue hacer una buena predicción al reconocer imágenes cuando aparece el tórax y la patas del cangrejo juntas, siempre y cuando las patas destaquen con respecto al tórax, mientras que las imágenes donde apenas se distingue la forma del cangrejo o cuando el tórax no está no se clasifican correctamente.

Aparte de los aciertos conseguidos para esta muestra, hay 6 coincidencias adicionales, que son erróneas. Buscando en el *dataset*, se han encontrado las imágenes de la **Figura 5.2**. La conclusión que se puede extraer de las tres primeras imágenes de consulta, de clase perro 3 dálmata, siendo poco evidente en el primer caso las formas que se pueden ver en la imagen de muestra y la pose de los perros en la segunda y la tercera, donde las patas y la cabeza recuerdan las formas del cangrejo, confunden al programa y las clasifica como clase cangrejo.

Figura 5.2: *Imágenes de consulta clasificadas erróneamente como clase cangrejo.*

En la cuarta y la quinta imagen, de clase microorganismo, habrá que suponer que, en la primera de ellas la forma alargada del microorganismo central se asemeja a una pata del cangrejo, el microorganismo en forma de hoz al cuerpo y, junto con el microorganismo del centro, el de la esquina inferior derecha han confundido a nuestro programa, ya que las formas de la imagen recuerdan a la de un cangrejo con su tórax y sus patas. La siguiente imagen de clase microorganismo tiene forma alargada y achatada, que podría asemejarse al tórax del cangrejo, pero queda muy lejos de la forma de la imagen de muestra, siendo una imagen algo difusa, sin poder encontrar visualmente los trazos principales como en la primera imagen error. En la sexta y última imagen de consulta error, de clase perro 1 labrador, concluimos que, por la pose del perro, los mapas de características han tomado unos valores hasta acabar clasificándose como clase cangrejo, reconociendo tórax y patas.

Se finaliza con esta primera prueba comprobando por qué para la siguiente clase, clase perro 1 labrador, solo se encuentran 3 aciertos. Es justo, el peor caso en este episodio mostradas en la **Figura 5.3**. Los aciertos obtenidos para esta clase son las Consultas 3, 9 y 12. Con esta imagen de muestra, los algoritmos reconocen una serie de patrones como son el hocico, los ojos, orejas y cráneo, ladeado hacia un lado, y además reconoce la lengua.

En el momento de comparar la imagen de muestra con las imágenes de consulta, resulta casi determinante los detalles de la lengua y que esté ladeado y perjudica el reconocimiento de imágenes de la misma clase, ya que, las patas, la cola o la forma del cuerpo quedan excluidas, cuando son componentes esenciales. Por lo tanto, en esta interacción, la imagen de muestra solo dificultará el reconocimiento de imágenes y los resultados finales, sabiendo que es determinante.

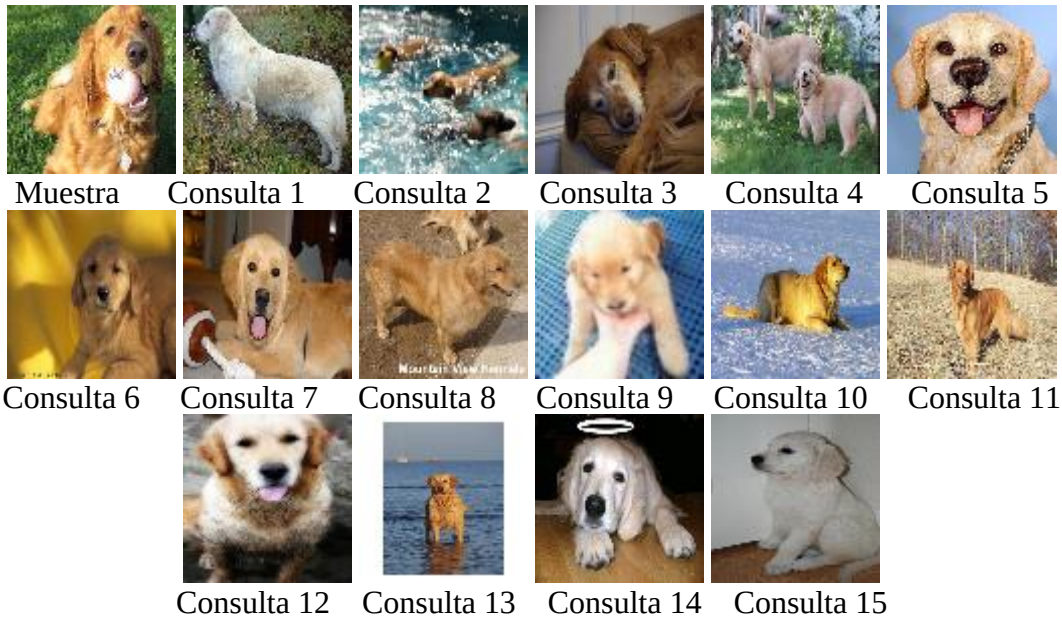

Figura 5.3: *Interacción de una imagen de muestra con imágenes de consulta de clase perro 1 labrador.*

Si el programa hubiese elegido una imagen de muestra con los trazos principales del perro, que serían la silueta del cuerpo, la cabeza más el hocico y las patas, habría reconocido más imágenes de consulta con su clase, pero tampoco demasiadas, ya que estos presentan una gran variedad de poses.

A continuación, se analizan los errores que comete el programa, clasificando imágenes como clase perro 1 labrador que se indican en la **Figura 5.4**. Esta clasificación puede verse algo confusa. Para la imagen consulta de la clase perro 2 husky se ven bastantes similitudes, como son la cabeza, hocico y lengua, pero a partir de esta imagen se vuelve todo un poco caótico. En la cuarta imagen de consulta, de clase cangrejo, se asemeja en que el tórax esta ladeado y de forma alargada, en la pinza izquierda se puede ver la forma característica de la oreja del perro de la imagen de muestra, que resulta poco visible. Para las dos primeras imágenes, de clase cangrejo, la carne de las patas puede asemejarse a la lengua del labrador en la imagen de muestra, pero en el resto de los casos se ha de suponer que ciertos trazos extraídos en el análisis de la imagen comparten semejanzas con la imagen de muestra que confunden al programa.

Figura 5.4: *Imágenes de consulta clasificadas erróneamente como clase labrador.*

5.2.2 Resultados de las pruebas para el dataset "Edificios UAM"

En este apartado se analizan los resultados del *datasets* "Edificios UAM", para el cual el programa obtiene los mejores resultados. Se analizará en primera instancia el modelo de una imagen de muestra, y posteriormente de cinco imágenes de muestra. Analizando los resultados del último episodio, se obtuvo un porcentaje de acumulación de acierto del 58.787%.

Aquí también se parte del mismo análisis hecho que en el anterior, viendo cómo funciona el programa para cada clase y su imagen de muestra. El análisis comienza por la clase que peores resultados tiene, vista en la **Figura 5.5.**

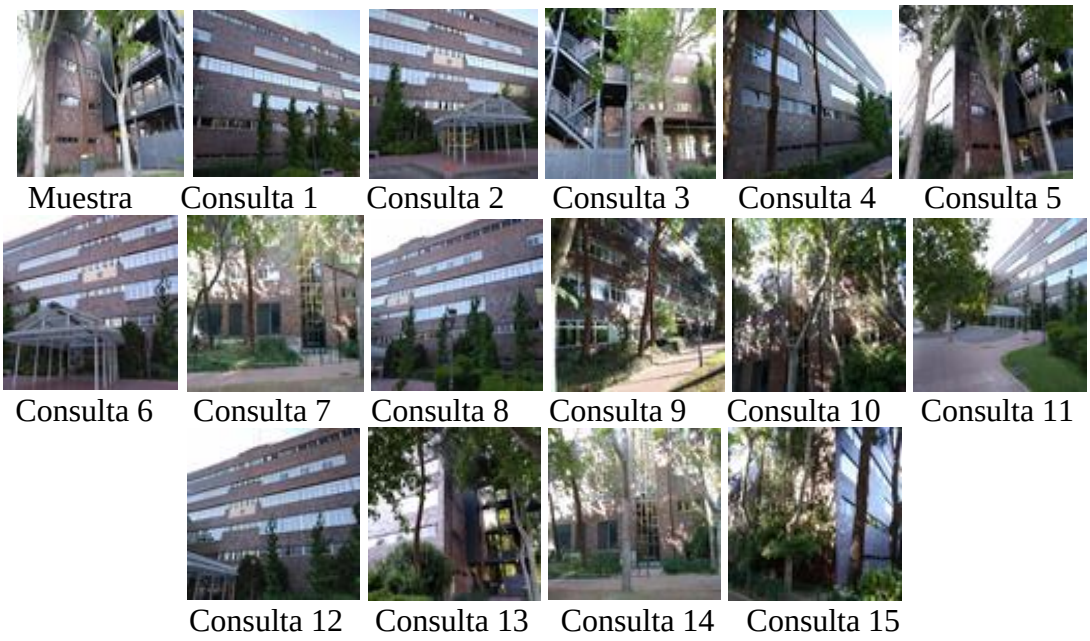

Figura 5.5: *Interacción de una imagen de muestra con imágenes de consulta de clase rectorado.*

Los aciertos obtenidos para esta clase, que corresponden a las de la clase de peores resultados con este *dataset*, son las Consultas 2, 3, 5, 7, 13 y 14.

En este caso, para la clase rectorado, cabe destacar que la imagen de muestra cuenta con diversos componentes que limitan bastante el reconocimiento de imágenes de consulta en la misma clase. Esto es debido a que en la misma imagen se encuentran dos árboles con troncos largos, la estructura de las escaleras exteriores y la parte del edificio principal destacando unas pocas ventanas.

En los aciertos se destacan la coincidencia de varios componentes, salvo en el caso de la Consulta 2 frente a la Consulta 6, difieren, ya que están tomadas desde diferentes ángulos. Habrá que suponer que reconoce en la fachada del edificio las ventanas, también el árbol en su forma alargada y algunos trazos más, porque los trazos que forman el porche de la entrada al edificio difieren mucho de la estructura de las escaleras externas, siendo este caso el menos intuitivo. Por lo general, las imágenes en las que más se destaca la fachada del edificio quedan fuera debido a esos componentes de la imagen de muestra.

A continuación, se muestran las imágenes que no pertenecen a la misma clase rectorado pero que son clasificadas erróneamente como tal, mostradas en la **Figura 5.6**.

Figura 5.6: *Imágenes de consulta clasificadas erróneamente como clase rectorado.*

Para estos casos, se ha de intentar ver la similitud con la imagen de muestra en la aparición de trazos rectos verticales, ya sea en los árboles o farolas de las cuatro primeras imágenes, pertenecientes a las clases posgrado y EPS que, como en la imagen de muestra, ocultan parte del edificio y en los trazos rectos oblicuos, y también los visibles en la forma de las

estructuras de las cuatro siguientes imágenes pertenecientes a la clase monumento. Para algunos de estos casos, los trazos de los monumentos se asemejan a todo el conjunto de las ventanas y el resto de los detalles de la imagen de muestra.

Acto seguido, se analizan las imágenes de la clase con mayor acierto, visibles en la **Figura 5.7**. Los aciertos obtenidos para esta clase son las Consultas 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12 y 13. En este caso, hay una tasa de acierto muy alta para esta clase plaza mayor, siendo una clase con imágenes bastante homogéneas. Los trazos están muy marcados, como los trazos oblicuos, horizontales y verticales de la silueta de las diferentes partes de la estructura, que presentan muchas similitudes.

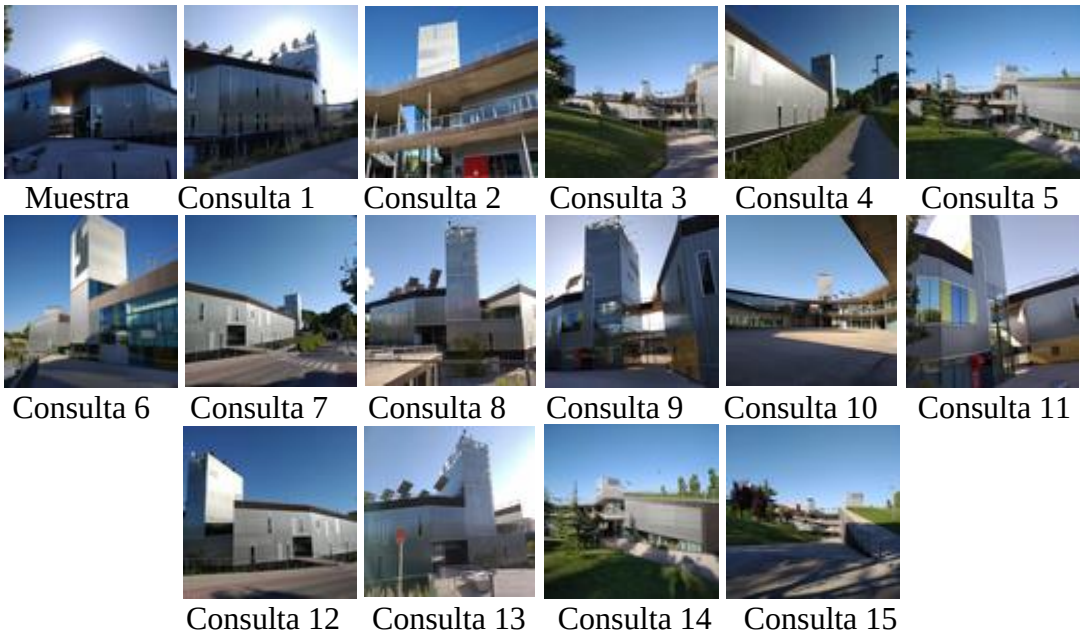

Figura 5.7: *Interacción de una imagen de muestra con imágenes de consulta de clase plaza mayor*.

De las imágenes mal clasificadas, la imagen Consulta 6 presenta muchas similitudes, pero no encuentra la conexión en los trazos que definen a la torre ya que, en esta imagen, se destaca sobre todo esta estructura frente al resto del edificio.

Las Consultas 14 y 15, pese a que son muy similares a Consulta 5, son clasificadas en distintas clases. La Consulta 14 es clasificada como clase monumento, cuando se analizan los trazos de las piedras montadas, se observa que la piedra tumbada podría asemejarse a la estructura principal del edificio y que la piedra que está de pie podría haber sido relacionada

con la torre que vista a lo lejos en Consulta 14. Por último, la Consulta 15 es clasificada como clase rectorado, por lo cual ha de concluirse que el juego de sombras confunde al clasificador y las características se aproximan a las de la estructura de la escalera externa, las ventanas, etc. No queda del todo clara esta clasificación. Una vez visto esto, se analizan las imágenes clasificadas erróneamente como clase plaza mayor de la **Figura 5.8**.

Figura 5.8: *Imágenes de consulta clasificadas erróneamente como clase plaza mayor.*

Se observa que hay ocho imágenes clasificadas erróneamente y ello resta calidad de clasificación a pesar de los numerosos aciertos. En las tres primeras imágenes de consulta, de clase rectorado, se destacan los trazos rectos de los tejados y las ventanas donde intuimos que estas se asemejan al hueco iluminado de la entrada a la plaza mayor que hay en la imagen de muestra.

Las cuatro imágenes siguientes, de clase EPS, destacan los trazos correspondientes a los tejados, a los trazos rectangulares de la ventana de la entrada del edificio que se ve oscura, como del cartel blanco que se puede ver en varias de las imágenes y como la cristalera de la tercera foto, que se asemejan al hueco antes mencionado de la imagen de muestra. De la última imagen de consulta, de clase monumento, destacan los trazos del hueco que forman las piedras montadas. Terminamos aquí con el análisis del mejor caso de predicción para el *dataset* de Edificios UAM con una imagen de muestra.

Para terminar este apartado, se analiza un caso de cinco imágenes de muestra. Como recordatorio, para cinco imágenes de muestra, se suman los mapas de características extraídos del *netEncoder* de cada una de las imágenes de muestra con sus homólogas antes de concatenarlas con las imágenes de consulta. Los resultados, vistos en la **Tabla 5.1**, son

mucho mejores que para el caso de una imagen de muestra. Con el mismo *dataset*, "Edificios UAM", en la última interacción se genera un resultado muy bueno, de 78.520% de acierto.

Se comienza con la clase de peor predicción expuesta en la **Figura 5.9**, de clase rectorado. Los aciertos obtenidos para esta clase son las Consultas 1, 2, 3, 4, 5, 6, 7, 9 y 13. Comparado con el caso peor para una imagen de muestra, presenta unas mejoras bastante apreciables. Al sumar los mapas de características de las cinco imágenes de consulta antes de la concatenación, se consiguen mayores probabilidades para que una imagen de consulta comparta más características que con solo una imagen de consulta. En este caso, suma las características de los trazos que pertenecen a las distintas ventanas, a las escaleras de frente, de los árboles, el tejado, etc.

Figura 5.9: *Interacción de una imagen de muestra con imágenes de consulta de clase plaza mayor*.

A continuación, se comprueban los casos donde falla el programa, como es el caso de Consulta 8, que es casi igual a Consulta 9, salvo en que está tomada desde el ángulo contrario. Además, ocurre algo similar en la Consulta 13 y se clasifican en distintas clases.

Posiblemente el fallo en esta clasificación está en que el detalle del tejado no consigue distinguirse y, por lo tanto, no consigue clasificarlas bien. Para Consulta 15, que es similar a Consulta 6, existe una situación parecida, donde las escaleras externas no presentan la misma disposición que en la imagen de Muestra 3. Para Consulta 14, que es similar a Consulta 6, es determinante la disposición de las escaleras. Por ejemplo, Consulta 10 tiene bastantes similitudes con Muestra 1 y 4, pero es determinante el que no haya trazos que señalen el tejado.

Ahora, se continua con las imágenes clasificadas por error como clase rectorado, como se observa en la **Figura 5.10**. Estas dos imágenes pertenecen a la clase monumento, clasificadas como rectorado. En la segunda imagen se ven grandes similitudes con la imagen Muestra 1, como es la disposición del monumento, los trazos oblicuos y verticales, pero en la primera se ha de suponer que cae en esta clasificación por una suma de mapas de características de las muestras. El modelo de cinco imágenes de muestra resulta óptimo al clasificar erróneamente solo dos imágenes. No se ha extraído ningún ejemplo con una imagen de muestra para esta clase, pero ha de suponerse que se disminuyen las imágenes erróneamente clasificadas comparando con el resto de los casos.

Figura 5.10: *Imágenes de consulta clasificadas erróneamente como clase rectorado.*

Para terminar con nuestro análisis, se pondrán de relieve los resultados para el caso mejor para cinco imágenes de muestra con el mismo *dataset*, vistas en la **Figura 5.11**, de clase plaza mayor. Los aciertos obtenidos para esta clase son las Consultas 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14 y 15. La mejora con respecto a una imagen de muestra es magistral. La suma de diferentes mapas de características es muy efectiva, tanto para el caso peor como para el caso mejor.

La Consulta 10, consultando los ejemplos expuestos antes, es un caso de fallo común y vuelve a ser clasificada como clase monumento. Por lo tanto, esta imagen presenta una dificultad considerable a la hora de clasificarse en su propia clase. Las imágenes de muestra

reúnen trazos del tejado, de la estructura en forma de torre desde distintos ángulos, de ventanas, de cristaleras, de las entradas, de los caminos y carreteras circundantes, etc.

Figura 5.11: *Interacción de una imagen de muestra con imágenes de consulta de clase plaza mayor.*

Se puede observar una mejora respecto a una imagen de muestra, cuando se comprueba que solo dos imágenes de consulta que pertenecen a otras clases son clasificadas erróneamente, para este caso como clase plaza mayor vistas en la **Figura 5.12,** como lo fue en el caso anterior.

Figura 5.12: *Imágenes de consulta clasificadas erróneamente como clase plaza mayor.*

Frente al modelo de una imagen de muestra, donde se tenían ocho imágenes fallidas, aquí solo hay dos. Por lo tanto, esta es otra muestra de buen funcionamiento para un modelo de pocas imágenes.

5.3 Conclusiones de las pruebas realizadas

En las secciones y apartados anteriores se ha demostrado que el programa tiene un peor funcionamiento para *datasets* de imágenes de entrada de trazos bien diferenciados, donde el objeto en común a analizar tenga algunas de las características siguientes: pueda ser una parte pequeña de la imagen, comparta trazos importantes con otras clases, cuando haya una gran diferencia en las formas de la misma clase o que estén tomadas desde diferentes ángulos, de manera que el programa no distinga los trazos principales y su relación.

En estas condiciones, al extraer los mapas de características en la RN, daríamos con un resultado relacional menos exacto de lo buscado.

Se ha comprobado que el cambio de los *datasets* afecta de forma relevante al resultado del programa, debido a que las características de las imágenes contenidas en cada *dataset* pueden variar mucho y algunas dificultan o facilitan un buen funcionamiento del programa.

Además, se han observado los cambios producidos al modificar el número de imágenes de muestra (*K-shots*). Estas imágenes, que actúan como imágenes de referencia con las cuales se han obtenido los trazos representativos de cada clase, condicionan el reconocimiento y la clasificación de las imágenes de consulta, que no son fijas.

En la **Tabla 5.2** se muestra una comparativa de los mejores y peores resultados obtenidos con los mostrados en la tabla Table 2 del artículo [1] de referencia. En esta tabla referenciada utilizan el *dataset miniImageNet* los modelos: *Matching Nets* [13], *Meta-Learn LSTM* [14], *MAML* [15], *Prototypical Nets* [16], *Relation Net* [1].

En la tabla se ha incluido con el nombre *TFG- niniImageNet1*, los peores datos obtenidos en las pruebas y con el nombre *TFG-Edificios UAM*, los mejores para comparar estos valores con los de los otros modelos.

Model	FT	5-way Acc.	
		1-shot	5-shot
Matching Nets	N	43.56 ± 0.84%	55.31 ± 0.73%
Meta-Learn LSTM	N	43.44 ± 0.77%	60.60 ± 0.71%
TFG-miniImagenet1	N	44.54 ± 0.23%	59.82 ± 0.34%
MAML	Y	48.70 ± 1.84%	63.11 ± 0.92%
Prototypical Nets	N	49.42 ± 0.78%	68.20 ± 0.66%
Relation Net	N	50.44 ± 0.82%	65.32 ± 0.70%
TFG-Edificios UAM	N	**58.46 ± 0.32%**	**78.03 ± 0.21%**

Tabla 5.2: *Comparación de nuestra RN, para los datasets con peor y mejor resultado, con estudios previos y el estudio seguido, remarcados en negrita los mejores casos, para 600 episodios en la fase de prueba.*

Empezando por los mejores resultados del programa, del *dataset* "Edificios UAM", estos supondrían una optimización en la clasificación de imágenes por clase cuando las particularidades de las imágenes son menos heterogéneas. En el caso peor, del *dataset* "*miniImageNet* 1", con imágenes de particularidades más complejas y usado en todas las investigaciones previas, podría concluirse que es el más objetivo al tener mayor variedad y riqueza de particularidades. Ese resultado quedaría dentro de nuestras expectativas y en una posición intermedia en la tabla de resultados, que ha de valorarse de forma positiva, aunque quede por debajo del resultado de la investigación seguida. Pero esto no quita que se deba tener en cuenta los dos casos, el peor y el mejor a la hora de sacar conclusiones, ya que pueden valer los dos casos según la comparativa que se quieran hacer.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este proyecto, se ha realizado un programa basado en el modelo RNA de CNN, para la elaboración de un algoritmo que permita la clasificación de imágenes por clase. Los resultados obtenidos por el programa al utilizar unos *dataset* de referencia han permitido identificar cuáles son las características de las imágenes para las que el algoritmo proporcionan los mejores y peores resultados.

Se ha presentado el origen de RNA, que dio lugar al perceptrón multicapa y la importancia en su funcionamiento de la unidad neuronal para la elaboración teórica de los mecanismos con el objetivo de resolver el problema a abordar y su realización práctica, que originaron a la CNN. Teniendo en cuenta cómo afecta el uso de pesos de entrada para formar sesgos iniciales apropiados a la RN, las operaciones y atributos de esta, el algoritmo de aprendizaje y las operaciones implicadas para llevarlo a cabo, etc. Debido al proceso de creación de la RN, se consigue un buen rendimiento del modelo CNN comparable a otros diseños más complejos de RNA.

El método CNN facilita la comprensión teórica y el desarrollo práctico de las redes neuronales y es una opción válida para el reconocimiento de imágenes en el campo de la investigación de la IA y las redes neuronales.

6.2 Trabajo futuro

En cualquier trabajo futuro, su realización ha de tener en cuenta cada uno de los modelos realizados por otros investigadores. De cada uno se pueden extraer conclusiones y mejoras oportunas.

Para intentar visualizar cambios en los resultados con perspectivas a evaluar este modelo en concreto, habría la posibilidad de probar a cambiar los distintos elementos que forman parte de la RN y añadir o modificar otros: desde la modificación más simple aumentando o disminuyendo el número de clases como el número de imágenes hasta cambios algo más complejos, como por ejemplo el número de filtros de la convolución, la tasa de aprendizaje, el uso de diferentes pesos iniciales; y mucho más complejos como implementar opciones alternativas a la retropropagación, variar la función de optimización y de estimación de error

para retropropagación, introducir ajuste fino, etc. Cada una de estas modificaciones supondría un cambio, tanto en el retoque de algunos de sus parámetros como en el de funcionalidades proporcionadas. En cada una de esas variaciones, se debería comprobar los resultados obtenidos y la relación coste y beneficio de cada propuesta.

Referencias

[1] F. Sung, Y. Yang, L. Zhang, T. Xiang, et al. Learning to Compare: Relation Network for Few-Shot Learning. In NIPS, 2018.

[2] M. Pradilla Rueda. Alan Turing, su obra y los efectos sobre la calculabilidad. D, RIMCI, vol. 1, n.º 2, ago. 2014.

[3] I. Pérez Verona, and L. Arco García. Una revisión sobre aprendizaje no supervisado de métricas de distancias. A brief review on unsupervised metric learning. Revista cubana de ciencias informáticas 10: 43-67, 2016.

[4] K. Shihab. A Backpropagation Neural Network for Computer Network Security. Journal of Computer Science 2 (9): 710-715, 2006.

[5] T. Liu, S. Fang, Y. Zhao, P. Wang, J. Zhang, et al. Implementation of Training Convolutional Neural Networks. In NIPS, 2015.

[6] Pytorch CNN. https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html#sphx-glr-beginner-blitz-neural-networks-tutorial-py

[7] GPU Nvidia. https://www.Nvidia.com/es-la/drivers/what-is-gpu-computing/

[8] CUDA – Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/CUDA

[9] Pytorch. https://pytorch.org/

[10] S. Ioffe, and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In NIPS, 2015.

[11] R. Vilalta, and Y. Drissi. A Perspective View and Survey of Meta-Learning. Artificial Intelligence Review (2002) 18. 10.1023/A:1019956318069.

[12] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, and K. Kavukcouglu. Matching networks for one shot learning. In NIPS, 2016.

[13] D. Kingma, and J. Ba. Adam: A method for stochastic optimization. In ICLR, 2015.

[14] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In ICML, 2017.

[15] S. Ravi, and H. Larochelle. Optimization as a model for few-shot learning. InICLR, 2017.

[16] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. In NIPS, 2017.

Glosario

TFG	Trabajo de Fin de Grado
ECM	Error Cuadrático Medio
RECM	Raiz del Error Cuadrático Medio
EMA	Error Medio Aritmético
RN	Relation Network
CNN	Convolutional Neural Network
RNA	Red Neuronal Artificial
RNN	Relational Neural Network
IA	Inteligencia Artificial
GPU	Graphics Processing Unit
CPU	Central Processing Unit
SGD	Stochastic Gradient Descent